UNIVERSIDAD AUTONOMA DE MADRID
ESCUELA POLITECNICA SUPERIOR

LEARNING HOW TO MODIFY TRAINING RATES IN SCENE-RECOGNITION CNNS.

Autor: Miguel Basarte Mena
Director: Marcos Escudero Viñolo
Supervisor: Jesus Bescos Cano

-MASTER THESIS-

Electronics Technology and Communications Department
Escuela Politecnica Superior
Universidad Autonoma de Madrid
September 2019

PÁZMÁNY PÉTER CATHOLIC UNIVERSITY
Faculty of Information Technology and Bionics

LEARNING HOW TO MODIFY TRAINING RATES IN SCENE-RECOGNITION CNNS.

Autor: Miguel Basarte Mena

Director: Marcos Escudero Viñolo
Supervisor: Jesus Bescos Cano

Video Processing and Understanding Lab
Informatics Engineering Department
Escuela Politecnica Superior
Universidad Autonoma de Madrid
September 2019

This work has been partially suported by Ministerio de Economia, Industria y Competitividad of the Spanish Government and Fondo Europeo para el Desarrollo Regional of the European Union under the project TEC2014-53176-R (HAVideo)

Resumen

En este Trabajo de Fin de Máster, se pretende encontrar una medida que permita modificar el valor de la tasa de aprendizaje de cada neurona individualmente en una red neuronal convolucional. Concretamente, nuestro objetivo es paliar los efectos del fenómeno conocido como *Olvido catastrófico*. A partir de una red entrenada para una tarea *fuente*, este concepto se refiere a la pérdida de rendimiento que sufre la red para la tarea *fuente*, cuando se entrena para una nueva tarea *objetivo*.

Para este objetivo, comenzamos adaptando una herramienta de visualización de redes neuronales para extraer conclusiones sobre el comportamiento y la actividad de las neuronas. Con esta información, planteamos la hipótesis de que las neuronas con mayor actividad a lo largo de las imágenes de un data-set pueden considerarse útiles para la tarea *fuente* y aquellas con menor actividad son propensas a ser tratadas como *espacio libre* de la red para aprender la tarea *objetivo*.

Para obtener una medida cuantitativa de esta actividad, aprovechamos la entropía de la distribución de las activaciones de las neuronas para diseñar funciones de ponderación que permitan adaptar dinámicamente la *tasa de aprendizaje* de cada neurona.

En la sección de evaluación, comparamos los resultados de estas funciones con una estrategia clásica conocida como *ajuste fino*, buscando obtener redes cuyo rendimiento conjunto para la tarea *fuente* y la tarea *objetivo* sea lo más cercano posible al rendimiento obtenido por dos redes diferentes completamente entrenadas para cada tarea.

Los resultados obtenidos sugieren que todas las funciones propuestas funcionan mejor que la estrategia de ajuste fino y algunas de ellas tienen un rendimiento cercano al paradigma de entrenamiento completo.

Palabras Clave

Red neuronal convolucional, tasa de aprendizaje, ajuste fino, aprendizaje por transferencia, olvido catastrófico, aprendizaje multi-tarea.

Abstract

In this Master's Thesis, we pretend to find a measure that allows modifying the value of the learning rate of each individual neuron in a convolutional neural network. Specifically, we aim to handle the effect of the phenomenon known as *Catastrophic Forgetting*. Starting from a network trained for a *source* task, this concept refers to the loss in performance that the network undergoes for the *source* task, when it is trained for a new *target* task.

To this aim, we begin by adapting a neural networks visualization tool to draw conclusions about the behavior and activity of neurons. Using this information, we hypothesize that those neurons with higher activity along the data-set images may be considered useful for the *source* task and those with lower activity are prone to be treated as *free space* for the network to learn the *target* task.

To quantitative account for this activity, we leverage on the entropy of the distribution of the neurons' activities to design weighting functions to dynamically adapt the *learning rate* of each neuron according to it.

In the evaluation section, we compare the results of these functions against a classical *fine-tuning* strategy focusing on obtaining networks whose joint performance for the *source* task and the *target* task is as close as possible to the performance obtained by two different networks fully-trained for each task separately.

Obtained results suggest that all of the proposed functions perform better than the fine-tuning strategy in this scope, and some of them perform close to the fully-training paradigm.

Keywords

Convolutional neuronal network, learning rate, fine-tuning, transfer learning, catastrophic forgetting, multi-task learning.

Agradecimientos

Quería comenzar este apartado de agradecimiento dando las gracias a mi tutor Marcos. Gracias por tu paciencia y tu buen hacer. Ha sido un placer volver a trabajar contigo como tutor.

También quiero agradecer a mis compañeros del VPU Elena y Alex. Han ayudado a hacer más amenos estos meses y además, me han ayudado siempre que lo he necesitado. Alex ha sido clave en la explicación de algunos conceptos que aplico en este TFM.

Además, quiero aprovechar para agradecer a mis compañeros de Máster estos dos años tan intensos. He descubierto a personas con un gran corazón.

Quiero remarcar también lo importante que han sido mis amigos del *Sótano de Manuela* aliviando mis tardes y llenándome de ilusión cada semana. No importa lo lejos que lleguemos con este proyecto, para mí ya ha merecido la pena.

Emi, Serra y Pabli, sois los mejores amigos que cualquier persona podría desear como compañía en los momentos buenos y en los momentos duros. Sabéis y habéis sabido estar cuando os he necesitado. Os quiero.

Mamá, Papá y hermanas, nuestra unión como familia se nota más cuando las cosas no son sencillas y ante el mayor reto que hemos tenido hasta ahora habeís mostrado que el amor todo lo puede. Mamá, eres la persona más fuerte que he conocido. Te quiero.

Por último,

Ju. Tú, que has sabido soportarme en los días que ni yo mismo lo hago. Tú, que has estado orgullosa de mí cuando nadie más lo estaba. Gracias por tanto amor.

A todos aquellos que me han acompañado a lo largo de este camino, os merecéis todo mi cariño y admiración.

Miguel, *2019.*

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Recently neural networks have become one of the most commonly used and most powerful tools in the field of machine learning. Specifically, convolutional neural networks have shown excellent performance applied to images for different and varied computer vision tasks, including scene segmentation or parsing, object and scene identification and recognition. However, for complex tasks, convolutional neural networks (CNNs) are required to be deep, i.e. made up of several thousands of parameters or weights, which are arranged into layers. These weights are usually set by posing the learning problem as a supervised one, requiring extensive human-annotated data. Due to the lack of publicly available data-sets needed for training, a common solution to this problem is the so-called fine-tuning strategy, where a network trained for a *source* task is tuned or adjusted to learn a new *target* task—which may or may not be related with the *source* task—. In this strategy, it is common to either freeze some of the network's initial layers or decrease their learning rate whereas allowing the last layers to specialize to the *target* task, throwing away some of the learning capabilities of the network architecture by under-using the architecture.

Furthermore, this strategy inhibits the creation of multi-task CNNs, leading to the phenomenon of catastrophic forgetting: as CNNs are adapted to the *target* task, performance for the *source* task is harmed. Lifelong learning (LLL) [1] and Learning without Forgetting (LwF) [2] are some of the learning strategies that have been proposed to cope with this problem. Both strategies are designed to adapt the network's learning rates to the relevance and the degree of use of the network's neurons. The concept of relevance is one of a subjective interpretation. However, it can be

somehow quantified as the impact of a given set of weights for a given task. This can be obtained by measuring the activation patterns —the set of neurons that are presented as more active for the *source* task indicate which are the most relevant neurons for this particular task and the least active ones can be understood as free space for learning the new *target* task—. Additionally, when fine-tuning a new task, one can measure the perturbance or modifications affecting the parameters learned for the relevant neurons, their weights, which we understand as their degradation.

In this project, as shown in the figure 1.1, we propose a method to deal with catastrophic forgetting. We start by adapting existing visualization tools [3] [4] to better understand the concepts of neurons' activation. Then, we will follow by designing and developing quantitative measure of this concept. Finally, we will incorporate these quantitative measures to adapt the learning rates for a *target* task, inspired by recent deep learning strategies [5].

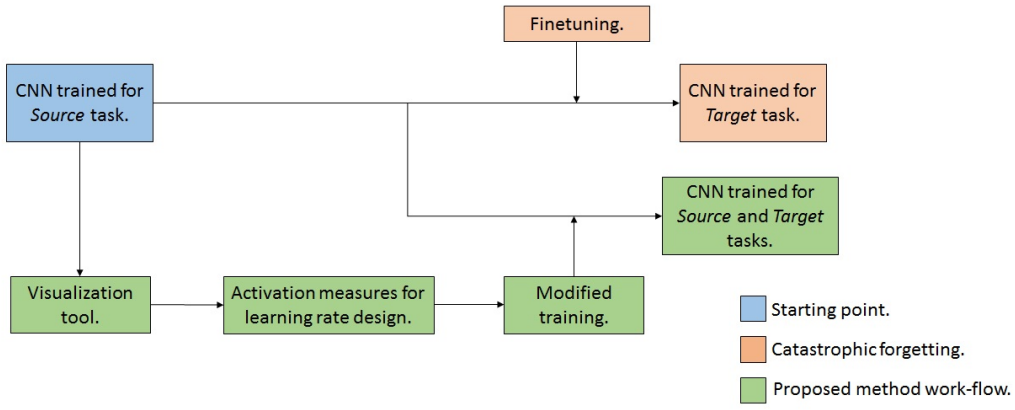


Figure 1.1: Diagram intro

1.2 Objectives

This master thesis seeks the next objectives:

1. Adapt existing visualization tools to the needs of this project.

2. Understand the neurons' activation patterns.

3. Design and develop a quantitative measure to estimate neurons' activation.

4. Design, develop and evaluate the effectivity of adapting the learning rates to the activations measures for the posed problem.

5. Document the project.

1.3 Document organization

This document is split in 5 chapters:

- Chapter 1: Introduction.
- Chapter 2: State of the art.
- Chapter 3: Design.
- Chapter 4: Experimental results.
- Chapter 5: Conclusions and future work.

Chapter 2

State of the Art

2.1 Introduction

Deep learning has become one of the main strategies in the area of machine learning and computer vision. This is due to the excellent performance they have shown for different and varied tasks such as scene recognition or objects classification. Different neuronal networks architectures exist in the literature, but in the field of image processing, the convolutional neuronal networks are the ones that work the best.

Although traditionally networks have been trained for a single task, in recent years new strategies have emerged that allow the use of a network already trained for a task—we will call it *source* task—to perform other tasks—called *target* tasks—. We will propose another strategy that take into account the activation and inhibition of the neurons presented in the network.

Even though the use of deep learning has many advantages, there exists also some drawbacks i.e. the need of a large amount of data to train the network, the vast number of parameters inside the network that need to be train—implies the need of powerful computers that works with GPU—and the great complexity that networks can achieve—making it difficult for human compression and leading to networks being treated as "black boxes"—.

In order to avoid this last problem, different strategies for visualizing parameters from a network are being proposed. In this work, we will use one of these proposals with the addition of some modules that we consider necessary to achieve the objectives that are pursued.

In this section we will explain these concepts in detail and we will show how recent publications dealt with the challenges presented.

2.2 Deep learning architectures

When working with neural networks there are many possible architectures to select, each with certain advantages and disadvantages over the others. The selection of one architecture over another is a key point in the design of applications based on deep learning and will depend on the final target application. Due to the fact that this work falls within the field of image processing, we will focus on the use of convolutional neural networks (CNNs) since they are the most applied for computer vision.

To understand how these CNNs work, it is essential to understand the operations occurring at each of the possible layers present:

- *Input layer:* This is the first layer in every CNNs and requires its input to be an image.
- *Convolutional layer:* The output of this layer will be the computation of the dot product between pixels values that are close in the input—this operation is called *convolution*—. Different parameters can be change in this layer i.e. size of the region where the convolution is calculated—knows as *kernel* size—or the overlapping between the regions—knows as *stride*—.
- *Pooling layer:* In order to reduce the dimensionality of the input, the pooling layer will apply a down-sampling operation along the spatial dimensions. This will allow making the network deeper without increasing the computational cost too much. Different strategies of sampling can be use in this type of layer such as *average-pooling* or *maximum-pooling*.
- *Activation layer:* This layer apply an element-wise activation function such as ReLu or Sigmoid function.
- *Fully-Connected layer:* In this type of layer all the neurons in the input are connected to all the neurons in the output. This operation results in a flattened matrix. Typically this layers appear at the end of the architecture.
- *SoftMax layer:* The last layer of the network adapt their value inputs to the number of classes that the task needs.

They exist many possible networks built from the combination of these layers. Some particular combinations have shown very good performance and they have receive their own name. Some of the most commonly used are: ResNet, VGGNet, Alexnet or Inception-v3
For the purpose of this thesis we have decide working with the Alexnet architecture

(Fig:2.1) trained over Imagenet [6] for different reasons i.e. the published works with which we want to compare the results of this research are based on this network, its use is very widespread so it is easy to find support on it and also, due to the research nature of this thesis, we seek to perform many and varied tests and Alexnet is not a very deep CNN, hence, easing the visualization and training tasks.

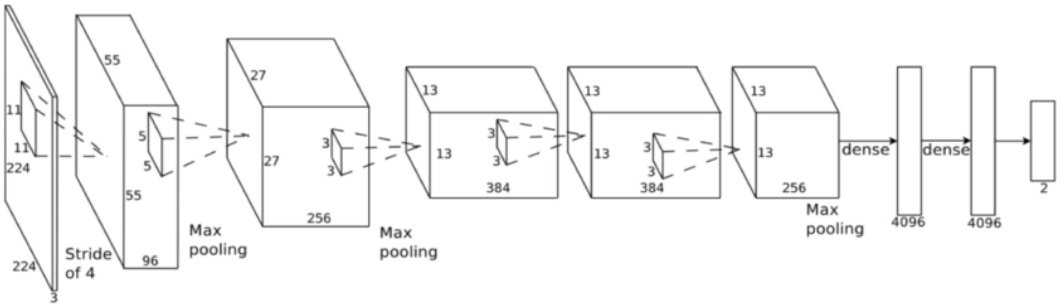


Figure 2.1: Alexnet architecture. Its consist in 5 convolutional layers, 3 max-pooling layers, 2 fully-connected layers and one soft-max layer.

2.3 CNN visualization

As we mentioned in the introduction and we are seeing along this section, CNNs can sometimes became highly complex. Depending of the network, the number of parameters can get to the order of millions making difficult the task of understanding why the network is or it is not working—thus subtracting capacity for optimize it—.

To cope with this issue, it is interesting to have a tool that allows us to visualize what is happening inside the CNNs. For this reason, this work aims to create a visualization interface by adapting an existing one in a way that is scalable and modular, allowing orderly access to information from all layers of the network and providing representation tools with independence of the network.

Next we show the different tools that have been studied and we expose the characteristics that lead us to accept or discard each of them.

2.3.1 Matlab

The program Matlab has several tools related to neural networks [4]:

- **Network Designer** With this tool we have a very visual approach for the development of networks, where we are allowed to vary the layers and some parameters of them (Fig:2.2).

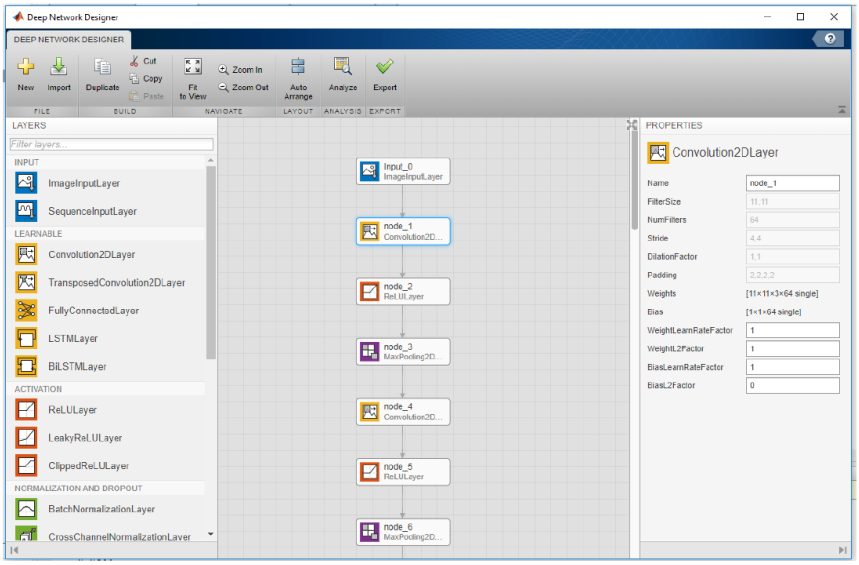


Figure 2.2: Matlab network designer

- **Network Analyzer** In this case, this tool allows us to visualize the structure of existing networks and provides us with a large amount of information on each of the layers of the network (Fig:2.3).

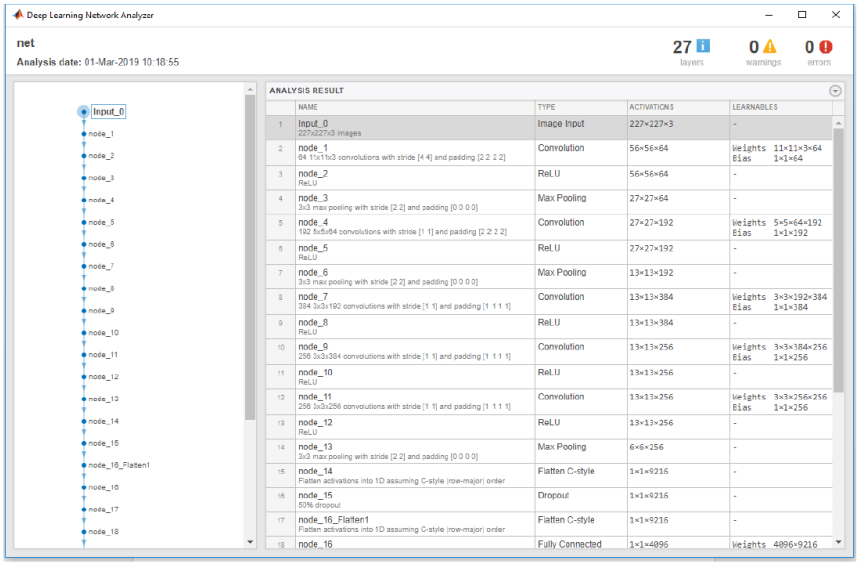


Figure 2.3: Matlab network analyzer

- **Weights Visualization** The visualization of the weights in Matlab is simple (Fig:2.4) but we find a problem: when the dimensions of the filters of a convolutional layer is greater than 3, its visualization and interpretation is not clear.

In this case, we opted for a PCA analysis to reduce the dimensionality to 3 and thus be able to represent it (Fig:2.5).

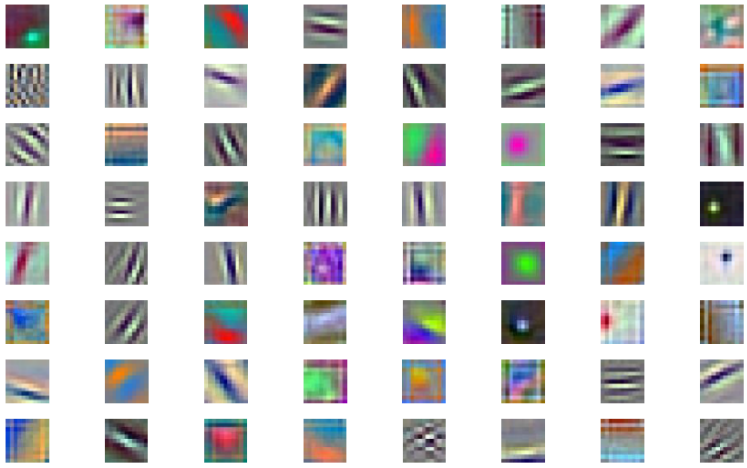


Figure 2.4: Matlab representation: Filters of the second layer of a CNN, with 3 dimensions.

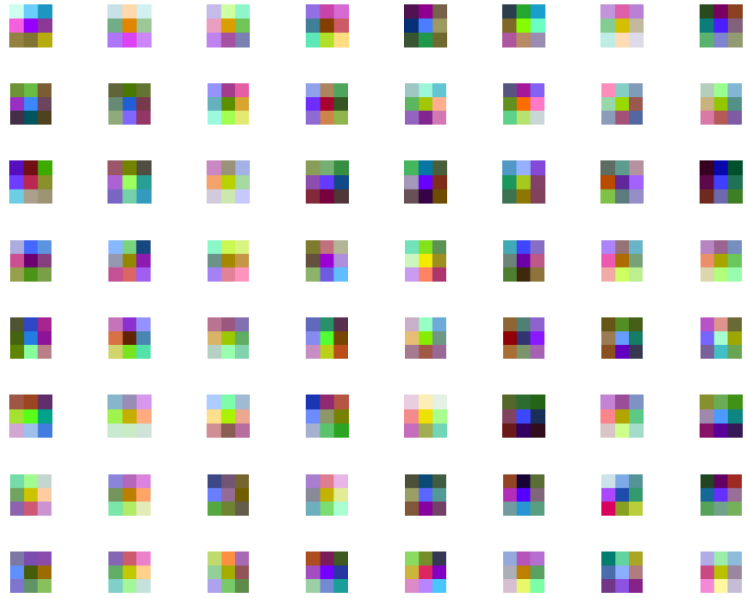


Figure 2.5: Matlab representation: Filters of the 8 layer of a CNN, after a PCA analysis.

- **Activations Visualization** The activations produced by the filters of the convolutional network when we introduce an input image can also be visualized in Matlab, as shown in the figure (Fig:2.6).

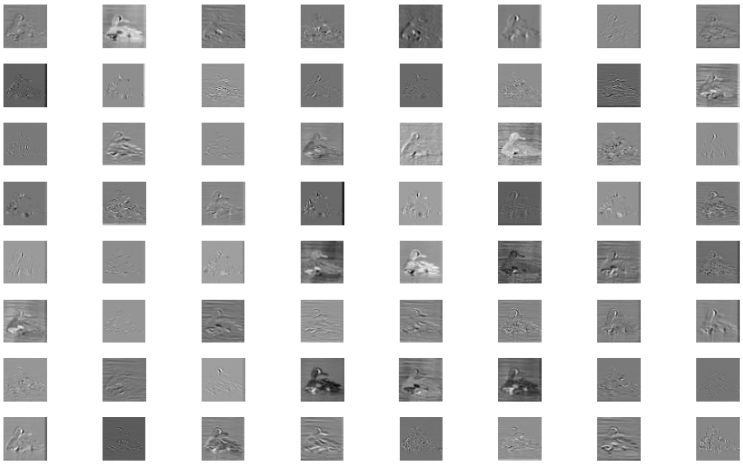


Figure 2.6: Matlab representation: Activations of a layer.

As can be observed, there are many benefits of using Matlab to achieve the objectives of this work but there are also some drawbacks. The most important one is the format in which the networks that work in Matlab are written: the vast majority of networks are created in formats such as pytorch or tensorflow and these are not formats that can be run in Matlab.

To solve this problem, we have studied the ONNX format [7] which allows us to change between the languages in which the networks are developed (Fig:2.7). By using this tool we have been able to translate some existing networks to be executed in Matlab but we have encountered another problem: Matlab does not have implemented some layers of convolutional networks in its current version (Matlab 2018b) so we have decided not using Matlab for this work.

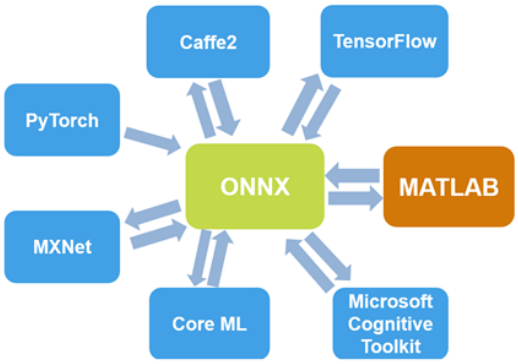


Figure 2.7: Possibilites with the ONNX format.

2.3.2 VisualDL

The next posibility we have studied is VisualDL [3]. This is a good tool for the visualization of scalars and histograms throughout the training of a network as well as the structure of the network (Fig:2.8), but the visualization of activations and weights is not implemented. For this reason, we decided to look for other alternatives.

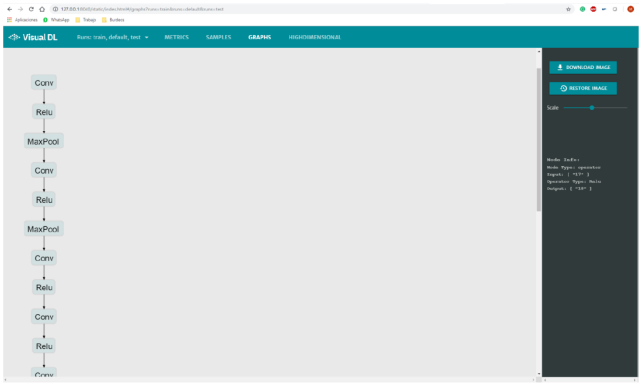


Figure 2.8: VisualDL interface.

2.3.3 Adam W.Harley, convolutional network visualization

In this case we have found a tool [8] that gives us a very clear visualization of a network trained with the MNIST [9] database. The resulting images are similar to those that are sought with this work (Fig:2.9) but the problem we find is that the tool is not very flexible and introducing a different network is a difficult task. Because we want our tool to work for different networks, we have decided to look for other solutions.

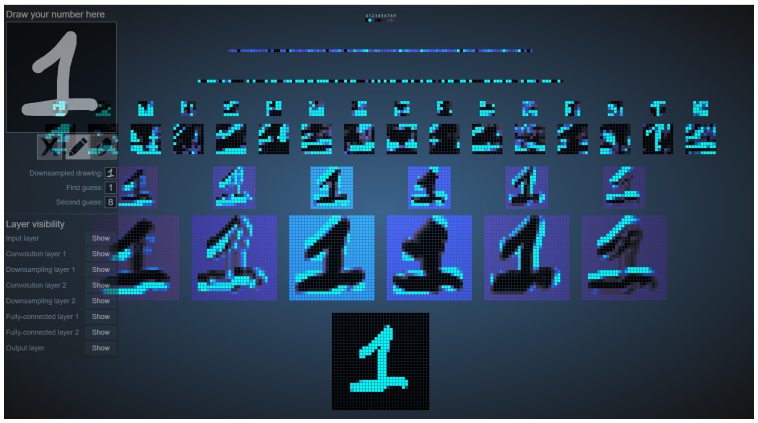


Figure 2.9: Adam W.Harley tool interface.

2.3.4 DeepVis Toolbox

The DeepVis Toolbox is the the last tool studied. The original version of the tool can be found in [10] and a more updated version of it is available at [11].

It is a very complete tool and we find it very suitable for the objectives pursued in this work. Therefore, the work focuses on adapting this tool and adding modules to it. In the section 3.1 we explain the main functionalities and disadvantages of the tool as well as showing an example of the interface (figure:3.1).

2.4 Deep learning training procedures

The concept of training a CNNs refers to a methodology that seeks to end with a CNN that has been taught to perform a specific task. For example, if we want to train a network in scene recognition we start by introducing to the network labelled scene data and the CNN will estimate an output (when the network is in the early stages of learning this output will be random but it will be getting closer to the correct output as the network learns). As the data is labeled, one can measure whether the network fails in its prediction and by how much. Internal weights of the network can be modified so the calculated output gets closer to the labeled output. This updating process is known as *back-propagation* [12].

An important parameter derived from the training of a CNN is the *Learning rate*. With the value of this variable we can select the impact that the *back-propagation* update has in the parameters of the network. A higher learning rate implies higher changes in the weights. Typically, this parameter is selected by the user and it will depend in several factors. In this work, we present a way of selecting the value of the learning rate automatically as a function of the relevance that each neuron of the CNN has for the trained task.

2.4.1 Scratch vs Transfer learning

When training a CNN for a task we are presented with two main alternatives:

- Training from scratch: The weights of the network are initialize randomly and the CNN is trained from there. It usually requires a very large amount of labelled images in the training phase in order to obtain an acceptable performance.

- Transfer learning [13]: This technique, also knows as fine-tuning, consist in initializing the network's weights from an already trained network in one task —*source* task— and then replacing the last layer in order to make it consistent

with the number of classes needed for the new task—*target* task—. Finally, the CNN is trained for the *target* task. When training, it is possible to freeze some layers or weights i.e. decrease their learning rate value. With this method the network is prove to undergo a problem (see section 2.5).

With transfer learning the amount of labelled images required as training data is lowered because the changes that are necessary in the majority of the weights to obtain a good performance are less than in the case when we train from scratch. The optimal number of layers to be frozen depends on the size of the training data-set and on the nature of the *target* task. If the *source* task is similar to the *target* task, freezing a large number of layers will give a good result as it is expected that their weights may be also similar. If both tasks are different, the network must be given more freedom to adapt to the new task i.e. larger learning rates.

2.4.2 Task relationships

As we mentioned, the relationship between two tasks is relevant when training a network for a new task. Training from scratch two similar tasks ignores their quantifiably useful relationships leading to a massive labeled data requirement. Alternatively, a model aware of the relationships among tasks demands less supervision, uses less computation, and behaves in more predictable ways. These relationships are not always trivial but recent studies [14] shows that it possible to design methods for obtaining relations—automatically—that specifies which tasks supply useful information to another, and by how much.

2.5 Strategies to avoid catastrophic forgetting

The strategy followed by the transfer-learning method works very well when it comes to learning a new task but has a problem: the *source* task for which the network was initially trained is forgotten. This is called *catastrophic forgetting* and in the last years different techniques have been developed that seek to avoid this problematic i.e. in addition to seek for good results with the *target* task, also focus on not losing performance in the *source* task.

The first approach [15] consist in learning, for each task, an under-complete auto-encoder that captures the characteristics that are crucial for the *source* task. Later, when a new task needs to be learn, preventing the auto-encoders from changing preserve the information on which the *source* tasks are mainly relying but at the same time giving the *target* task enough space to be learned.

Other way to use auto-encoders to prevent *catastrophic forgetting*—as proposed in [16]—is to use them to learn the basic characteristics that images from a task share and having different experts in the same system trained for the different tasks. This way when a new image is presented to the system these auto-encoders will work as a gate that decide to which expert forward the image.

Last strategy studied [17] consist in penalizing the changes to the network parameters in the *target* task training stage depending on their importance and knowledge on the *source* tasks. This penalization is delivered by modifying the learning rate of the neuron: low when the parameter is consider important and higher when the parameter is less important. To measure the importance of the parameter, the study proposes taking into account the magnitude of the gradient of each parameter when the network is being trained for the *target* task. Our method follows this research line but changing how the importance of a parameter is defined: we determine the importance of a parameter as a function of its activation pattern when images to the network are introduced(further explanation in section 3.4).

Chapter 3

Design

In this section we define and motivate the proposed method. We begin with an explanation of the visualization tool. Next, neurons activation patterns are explained as well as their distributions. We also explain the design of a measure to estimate neurons activity along a data-set. Last, we explain how to apply this knowledge to the individual learning rates of a network in order to reduce catastrophic forgetting (phenomenon explained in 2.5).

3.1 CNNs visualization tool

As the title of this master thesis suggests, the idea of this work is to find different ways to adapt the learning rate of a convolutional neuronal network focusing on preventing catastrophic forgetting in the *source* task when training the CNN for a new *target* task.

The first step to achieve this objective is having a tool that allow us to observe how the neurons of the CNN react when an input image is presented. This would allow us to draw conclusions on how the neurons behave and to discover the patterns that these neurons have learned to recognize.

As explained in section 2.3.4 we have decided to start with the DeepVis Toolbox and add some modifications in order to adapt it to our needs. The basic layout of the visualizer can be seen in Figure: 3.1. In the top left corner we observe the input image. We can select between all the images of the data-set to visualize how the others modules change depending on the input. It is also possible to visualize any layer of the CNN changing the selected layer in the top center of the tool. Right in the middle of the layout we have all the neurons of the layer selected (in this case we have selected the first convolutional layer of an Alexnet network [6] trained with

the Indoor scene recognition data-set [18]). We can move along the neurons to study their particular behaviour.

The image at the right side represents the maximal activations of the neuron selected. It is a composition of 9 patches calculated offline over a data-set by cropping the zones of the images that result in the highest activations for each neuron. This give us a clear representation of what are the patterns that each neuron has learned to recognize—in this case, the neuron selected react with high activation when in the input image are presented changes from yellow to blue—.

Right below the input image we can see in detail the activation pattern of the neuron selected. As expected by the learned pattern, this neuron has a high activation in the zones where yellow or blue appear. Lastly, at the right side of the tool we can select between different options to change the visualization modules.

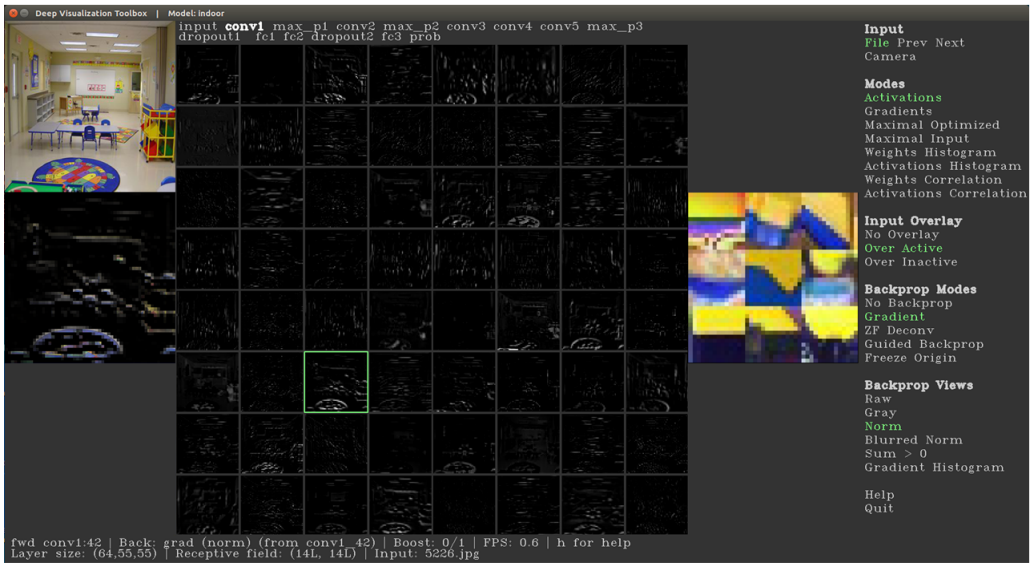


Figure 3.1: DeepVis Toolbox interface.

Even though this tool is at an advanced state of development point, it is not finished yet, so we have implemented some changes and new options that weren't in the original tool. For example, we have improved the calculation of the maximal activations for any network. Besides, we have included the possibility of introducing the mean and standard deviation of the data-set to normalize the outputs.

Although this tool fits our objectives, we have found a problem: the language in which this tool has been developed is Caffe[19] and the CNNs that we need to visualize are stored with a pytorch extension. To solve this problem, ONNX [7] was

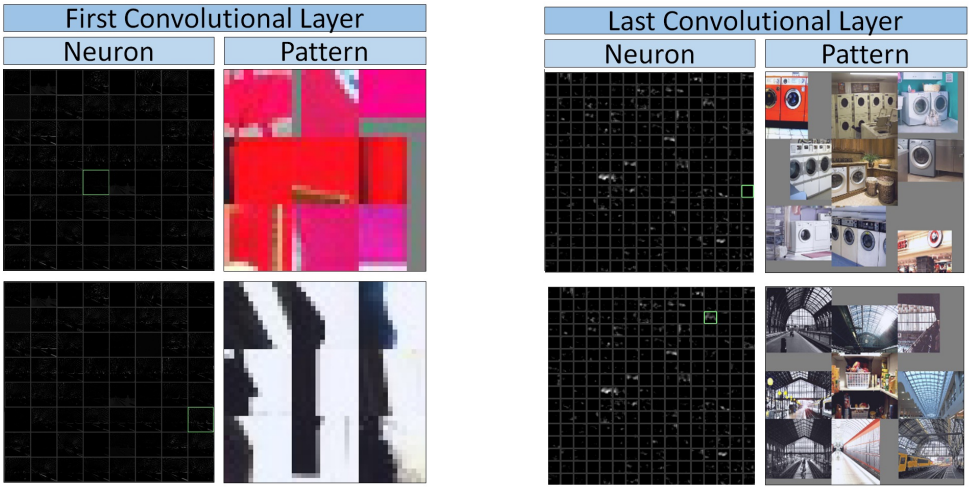
not an option since it does not provide support to Caffe (Fig:2.7), so we studied several alternatives and the one that works the best was PytorchToCaffe [20].

3.2 Activation patterns in CNNs

A CNN trained for a specific task implies that some neurons of the network have learned to recognize patterns in the images. If that pattern is present in the input image, the neuron will have a high activation, if not, the neuron will show low activity.

The patterns recognized by the firsts convolutional layers are usually simple, e.g. colors or simple shapes. It is the connection between these patterns what permits recognition of more complex patterns by the network as we move deeper in the CNN.

Using the visualization tool we can observe what patterns each neuron has learned to recognize. In the figure 3.2 we represent 4 different neurons (marked in green) and their learned patterns. These neurons belong to a network with Alexnet architecture [6] trained with the Indoor scene recognition data-set [18]. The firsts two neurons (Fig: 3.2a) are extracted from the first convolutional layer; the other two (Fig: 3.2b), belong to the last convolutional layer of the network.



(a) First convolutional layer. (b) Last convolutional layer.

Figure 3.2: 4 different neurons and their learned patterns.

In the first figure (3.2a), the first neuron has learned to recognize the color red and the second one detects abrupt changes from black to white (it can be understood as a border detector). This means that if the input image contains something colored in red, the first neuron will show a high level activation in the zones where the color appears. It will happen the same for the second neuron when borders of similar

direction and width are included in the input image.

In the second figure (3.2b), the connection between the neurons in the previous layers allows the last convolutional layer to learn complex patterns such as washing-machines or glass-windows. These neurons will show high activity when such structures appear in the input image.

This representation confirms what we stated before: the simplicity of the patterns in the firsts layers and the increase in complexity as we move deeper in the net. When training for a new task, it is expected that the first layers won't need to change a lot since the patterns are likely to be common between tasks because they are not task-specific. But the last layers are prone to change as their patterns are characteristic of the current task, i.e. the detection of the color red is useful for a lot of different tasks but the recognition of washing-machines is only useful for some specific tasks.

3.3 Distributions of neurons activations

By the use of the visualization tool on different networks, we have found that some neurons are activated with higher level and with higher frequency along the data-set. This phenomenon can be explained by the nature of the pattern that each neuron has learned to recognize, because some of these patterns are more common than others.

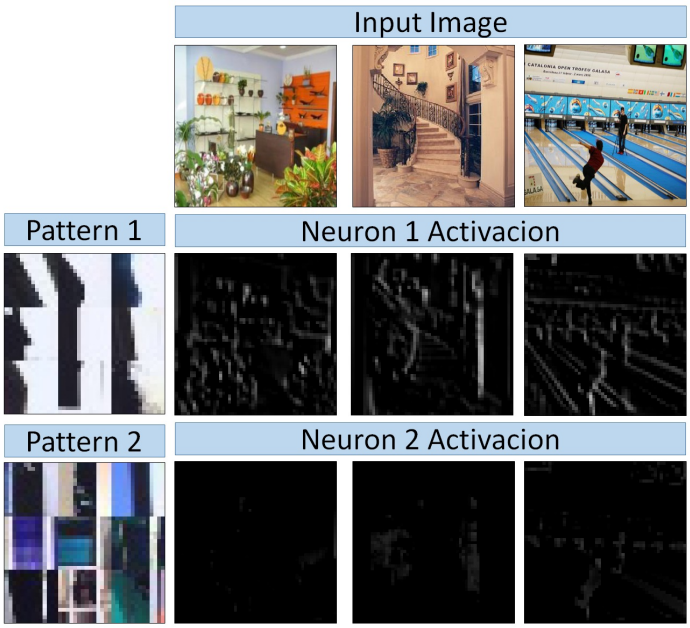


Figure 3.3: Activations from 2 different neurons over 3 images.

With this information, we formulate our hypothesis: the neurons that have higher activations (in level and in frequency) are more useful for the task the network is trained for, than those neurons that are almost never activated. Therefore, neurons that appear to be less useful are going to be treated as *free space* i.e. areas that can be used for learning new tasks. Differently, changes on highly activated neurons are to be restricted, to avoid a decrease in the performance of the CNN for the *source* task.

Examples of activations shown by two different neurons can be observed in figure 3.3. Like other examples shown in this chapter, these neurons belong to the first convolutional layer of an Alexnet architecture [6] trained with the Indoor scene recognition data-set [18]. For all the 3 images, first neuron presents high activation while the second one presents low activation. This can be explained by the pattern they learned; the first pattern is more common than the second one. In this case, our hypothesis defends that the first neuron has a high relevance for the scene recognition task while the second one provides little information.

3.4 Activation measure

The activity shown by a neuron over different images gives information about the relevance of that neuron for the task. Therefore, we are interested in designing a measure that allows us to estimate this activation and use it to design individual learning rates for each neuron of the CNN.

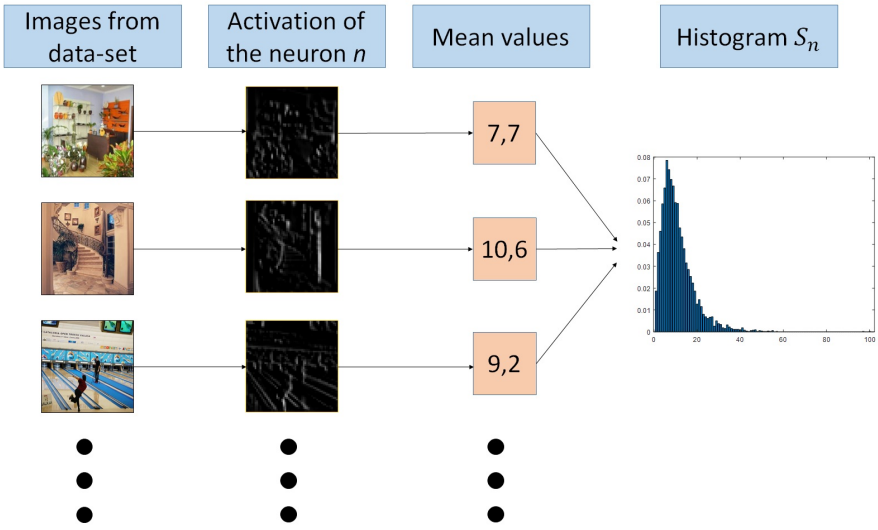


Figure 3.4: Design process of the activation measure.

In order to explain how the design of this measure was done, we will focus on explaining how to obtain this measure for a particular neuron n. As can be observed in figure 3.4 we start by calculating the activation images from the neuron n over the data-set images. Later, we obtain the mean value of each activation image in order to build the histogram S_n.

Each histogram S_n represents the mean value distribution of the neuron n along the data-set. In the figure 3.5 two different histograms can be observed. The first one, belonging to the neuron n, shows a high dispersion in the mean values while the second one, belonging to the neuron n', shows average values more compacted and close to zero. Being consistent with our hypothesis, we consider the neuron n as a useful neuron while neuron n' is prone to be considered as *free space*.

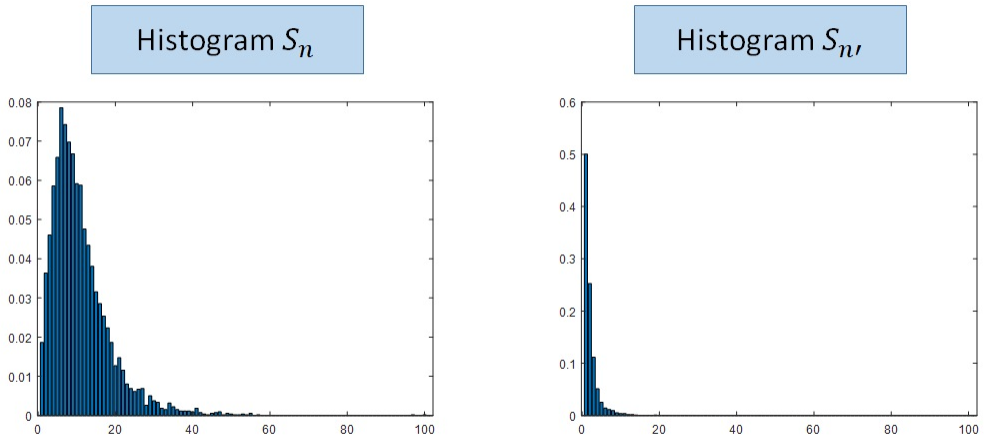


Figure 3.5: Two different histograms: S_n and $S_{n'}$

If we now calculate the entropy of the histogram S_n (equation: 3.1), we obtain an indicator of how dispersed is the distribution. We will refer to this value as H.

$$H = -\sum S_n log_2(S_n) \tag{3.1}$$

Those neurons that are relevant to the task will have a high H value. However, those neurons that do not contribute much to the task, will have a low H value.

3.5 A dynamic learning rate

The last step of the proposed method is to use the H value of each neuron to design an individual learning rate according to its relevance in the network. We understand that a high value of H in a neuron implies that it needs a low learning rate if we want to maintain its relevance for the *source* task. A low H value implies that the neuron should be assigned a high learning rate in order to *free* that neuron so it can adapt to the *target* task.

Different functions that meet the above can be designed. In section 4.3.3 we motivate the selection of some of these functions and we discuss the results obtained with them.

Chapter 4

Experimental results

4.1 Aim

The proposed experiments are designed to validate the hypothesis of this project i.e. the catastrophic forgetting problem can be reduced by designing a strategy to change the values of learning rates according to the entropy of the distribution of the activations show by the neurons for the complete *source* task data-set (as mentioned in section 3.4, we refer to this parameter as H).

It is intended to achieve a CNN that has been successfully trained for a *target* task without loosing performance in the *source* task.

4.2 Explored data-sets

The selection of the data-sets and tasks for this work is motivated by recent works [17] related to the posed problem. We have selected 2 different tasks with their associated data-sets.

First task is about indoor scene recognition. With 67 different indoor scenes categories the CNN has to be able to distinguish which category each image belongs to. The data-set associated contains 6700 labelled images from indoor scenes, where 5360 images are marked for training purposes and 1340 for testing. We can observe some samples from the data-set in the figure 4.1.

Figure 4.1: Samples from the indoor scene data-set. From left to right: *gym*, *warehouse* and *office*

The second task is defined as flowers species recognition. This time, the CNN has to be able to correctly predict whether the image belongs to a flower specie or another. The number of possible species is 102. The data-set associated with this task contains 8189 labelled images of flowers species distributed in 6149 images for training and 2040 images for testing. We observe some samples from this data-set in the figure 4.2.

Figure 4.2: Samples from the flowers data-set. From left to right: *hard-leaved pocket orchid*, *english marigold* and *canterbury bells*

4.3 Experimental setup

As mentioned in 2.2 we work with the Alexnet architecture as the main CNN for the proposed experiments. To achieve consistent results, the following parameters have been maintained throughout the experiments:

- **Learning rate**: We start by selecting a global learning rate of 0.008 for all the experiments. This global value applies to each neuron in the learning phase

but, when applying the method proposed, it is weighted by a coefficient (from 0 to 1) that depends on the value of H of each particular neuron. Definition of the learning rate concept can be found at section 2.4.

- **Epochs**: We train all the networks for 500 epochs and we select the one with better accuracy over the test data-set.

- **Decay value**: Best results were obtained with an *L2 regularization* with a *lambda* value of 0.0001. This regularization together with the data augmentation method, prevents the network to suffer from *overfitting*. *Overfitting* refers to a problem which occur when the CNN learns in detail the training data-set, negatively affecting the performance of the network on the testing phase.

- **Data augmentation**: Refers to a process where the training data available is transformed to significantly increase the diversity of data available for training models, without actually collecting new data. We decided to use two different data augmentation techniques for all the experiments: *random crop* and *random horizontal flip*.

- **Evaluation techniques**: Due to the fact that we use *random crop* and *random horizontal flip* in the data augmentation phase, we have decided to use *ten-crop* technique. This technique is used in the testing phase and consist in cropping the input image into 5 smaller images selecting the 4 corners and the center of the image. Then, we also include their reflection and consider the output of the CNN for these 10 images. Final result consists in an average between the outputs of these 10 images.

- **Number of images used in training and test**: During all the experiments we have used the division specified by the data-set for training and testing. This means:

  ✧ Flowers data-set: 6149 images for training and 2040 for testing.

  ✧ Indoor scene data-set: 5360 images for training and 1340 for testing.

4.3.1 Transfer learning from basic Alexnet

The baseline of this project consist in two different CNNs: one trained for indoor scene recognition and the other one for flowers recognition. Both were obtained by transfer learning (technique explained in section 2.4.1) from an Alexnet network trained over Imagenet data-set.

For this work, the accuracy of these two networks over their test data-set is understood as the highest possible. It is treated as the optimal value to which our networks should tend. Our final goal is to have a single network that can perform both tasks simultaneously with results as close as possible to these operational limits defined by these two networks.

Using the parameters established before, we obtain 62,61% accuracy for the indoor scene task and 85,88% accuracy for the flowers recognition task.

4.3.2 Fine-tuning strategy

Next step consists in applying fine-tuning method to the baseline CNNs. Starting with a network trained for a *source* task, we end with the same network but this time trained for the new *target* task. In this process, the learning rate remains the same for every neuron, no matter their relevance. This method shows clearly the catastrophic forgetting problem since, although a good result is achieved for the *target* task, the performance in the *source* task is greatly affected. This training process is design with the same parameters explained in the introduction of this section.

4.3.3 Proposed transfer functions

Based on the assumption that a greater entropy implies greater relevance of the neuron for the task, we designed functions inversely proportional to the value of H. Neurons that show high activity are prone to be assigned a small learning rate, limiting their plasticity to change. Differently, neurons with low activity are assigned a high learning rate; hence, allowing them to adapt to the new task.

We divide the transfer functions into two different groups attending to the way they handle intermediate values. Hard decision functions only allow the coefficient value to be 0 or 1. In the case of soft decision functions, the coefficient can vary between all the real values from 0 to 1.

For each neuron, their coefficient calculated with these functions is multiplied by the global learning rate. The result of this operation is the final learning rate applied to that neuron. Therefore, this method allows us to individually modify the effective learning rate for each neuron of the network.

4.3.3.1 Hard decision transfer functions

Hard decision functions allow neurons to adapt to the new task without any restriction when their H value is below a selected threshold. Neurons with H above the threshold are frozen, restricting any changes into them. We use 3 hard decision transfer

functions in the experiments.

Mathematical description of this type of functions can be observed in equation 4.1.

$$f_{1,2,3}(H) = \begin{cases} 0 & if \quad H > th \\ 1 & if \quad H < th \end{cases} \tag{4.1}$$

Where $f_{1,2,3}(H)$ reefers to each of the 3 different functions designed. These functions differ on the value of the threshold. Higher threshold implies more room for the network to adapt but at the cost of loosing performance in the *source* task. Because we want to reduce catastrophic forgetting we limit the flexibility of the CNN by decreasing the threshold value. The functions used in the experiments can be observed in the figure 4.3.

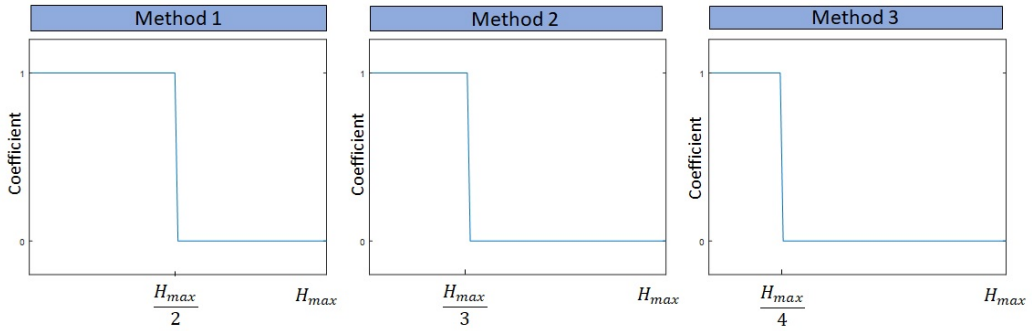


Figure 4.3: Hard decision transfer functions designed

4.3.3.2 Soft decision transfer functions

This type of function assigns a weight to every neuron in the CNN depending on their H value. It is possible to modulate the freedom of a network by changing the curvature and the slope of the functions. If the function has a large slope, the network will be better at maintaining its performance for the *source* task but losing capacity on adapting to the *target* task. A slower slope, implies better adaptability but greater loss in the initial task.

We have designed 3 soft decision functions following the equations: 4.2, 4.3, 4.4.

$$f_4(H) = 1 - \left(\frac{H}{H_{max}}\right)^2 \tag{4.2}$$

$$f_5(H) = 1 - \left(\frac{H}{H_{max}}\right) \tag{4.3}$$

$$f_6(H) = \exp\left(\frac{-2H}{H_{max}}\right) \tag{4.4}$$

Their graphical representation can be observe in the figure: 4.4.

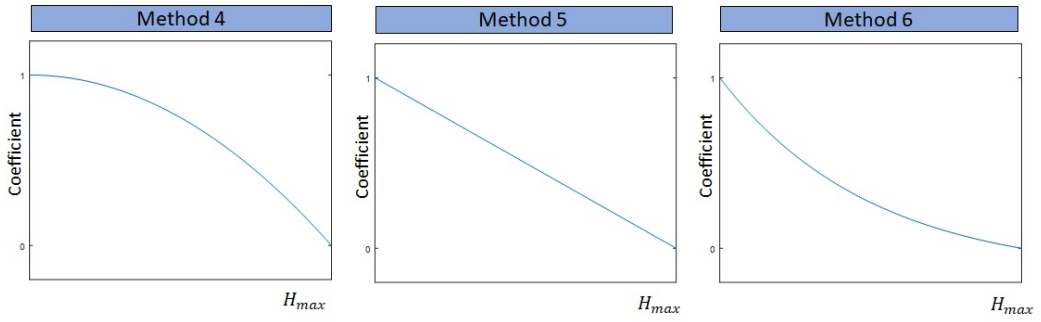


Figure 4.4: Soft decision transfer functions designed

4.4 Experimental results

The results presented here have been extracted from two different experiments consisting in starting with a CNN trained in a *source* task and training the network to learn a new *target* task. Is in the training phase when the methods explained in previous sections are applied.

Results of both experiments are shown with two different representations:

- **Numerical representation**: The accuracy obtained for all the methods in both experiments is displayed in the figures 4.5 and 4.7 respectively. Also, the accuracy loss compared with the baseline CNNs is calculated in the loss column. Cells marked in red show the existence of catastrophic forgetting and bold numbers mark the methods that obtain the best result for each task. Last, the right column shows the percentage of non-frozen neurons depending on the method.

- **Accuracy differences**: Visual representation of the loss column mentioned above can be observed in the figures 4.6 and 4.8 for each experiment respectively. Taking the optimal result as a 0% degradation, best method is the one that that results in the shortest bars.

4.4.1 From indoor to flowers

This experiment consists in using the indoor scene recognition task as the *source* task and the flowers species recognition as the *target* task. Then, all different methods are

applied and their results collected in figure 4.5

	Source task		Target task		
Indoor to Flowers	Indoor		Flowers		
	Accurracy	Loss	Accurracy	Loss	Free neurons
Baseline	62,61%		85,88%		100%
Finetuning	45,37%	-17,23%	82,28%	-3,59%	100%
Method 1	52,46%	-10,14%	81,10%	-4,78%	40,02%
Method 2	57,08%	-5,52%	80,40%	-5,48%	14,76%
Method 3	**57,16%**	**-5,44%**	78,48%	-7,39%	7,99%
Method 4	48,58%	-14,03%	**81,41%**	**-4,47%**	100%
Method 5	52,01%	-10,59%	81,29%	-4,58%	100%
Method 6	54,40%	-8,12%	80,68%	-5,20%	100%

Figure 4.5: Numerical results from the experiment indoor to flowers.

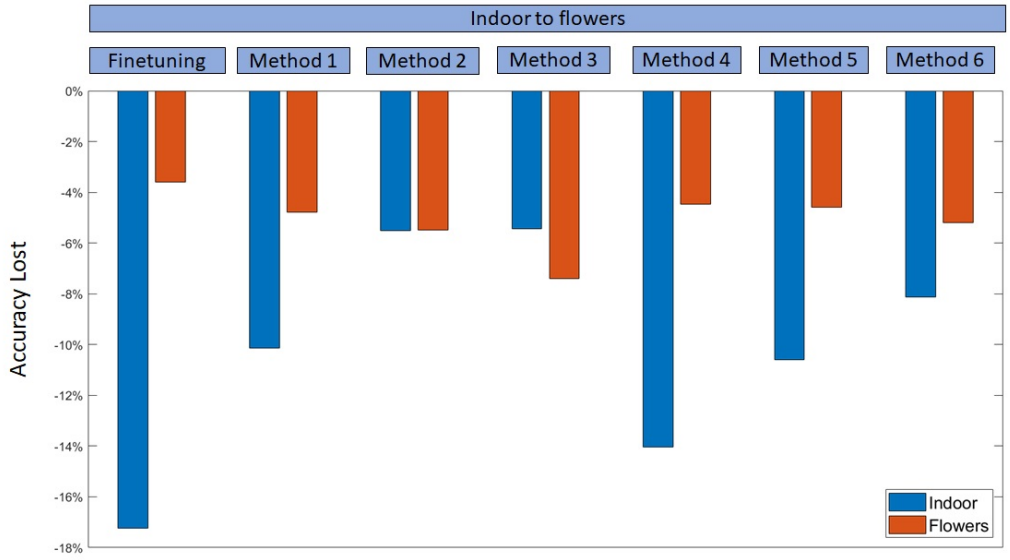


Figure 4.6: Accuracy differences between the baseline and each method for the experiment indoor to flowers.

4.4.2 From flowers to indoor

In this case, the flowers species recognition is established as the *source* task and the indoor scene recognition as the *target* task. Again, all methods are applied and the results are collected in figure 4.7

Flowers to Indoor	Source task		Target task		
	Flowers		Indoor		
	Accurracy	Loss	Accurracy	Loss	Free neurons
Baseline	85,88%		62,61%		100%
Finetuning	61,61%	-24,26%	60,82%	-1,79%	100%
Method 1	74,67%	-11,20%	59,47%	-3,13%	39,93%
Method 2	80,85%	-5,02%	58,73%	-3,88%	15,89%
Method 3	**82,38%**	**-3,49%**	56,71%	-5,89%	9,46%
Method 4	64,44%	-21,43%	**60,37%**	**-2,24%**	100%
Method 5	70,56%	-15,31%	60,12%	-2,49%	100%
Method 6	72,04%	-14,84%	59,70%	-2,91%	100%

Figure 4.7: Numerical results from the experiment flowers to indoor.

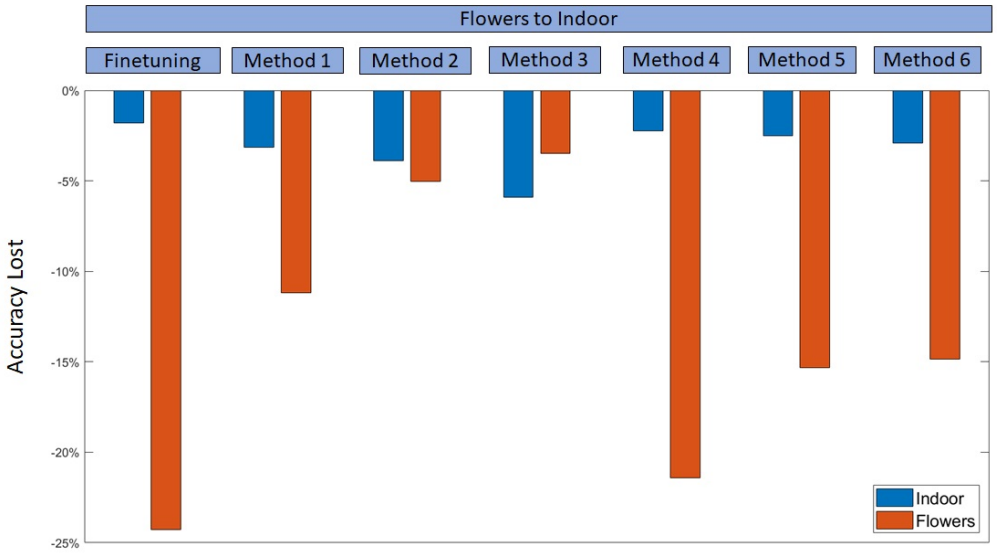


Figure 4.8: Accuracy differences between the baseline and each method for the experiment flowers to indoor.

4.5 Overall discussion

Both experiments show similar results when applying the same methods.

Catastrophic forgetting problem can be clearly observed for both experiments: performance in *source* task drops for 17% for the first experiment and 24% for the second one when fine-tuning method is applied.

4.5.1 Results when applying hard decision functions

To cope with catastrophic forgetting, we start by using the first designed function (method 1 in the figure 4.3). With this method around 40% of the neurons are unconstrained for both experiments. This first approach already shows a good result: the performance in the *target* task remains similar while the accuracy on the *target* task improves in more than 7% for the first experiment and around 13% for the second one.

Owing to the high performance of method 1 for the *target* task, which is close to the one obtain by the fine-tuning strategy, we decide to freeze a higher number of neurons. To this aim, we propose to decrease the threshold so that only around the 15% of the total neurons are unconstrained. With this second method, the improvement in the *source* task is remarkable: both experiments increase their performance in around 5% with respect to the method 1.

The performance decrease in the *target* task remains small (around 1% compared to method 1) in comparison with the higher increase in the *source* task; hence, we tried decreasing even more the threshold value. With the third method we obtain the best results for the *source* task but it shows a significant impact in the *target* task performance in both experiments. Only about the 8.99% of the neurons are re-trained for the *target* task and, in the light of the results, it is not enough for the CNN to adapt. Therefore, we decided to cease in decreasing the threshold value and look for another approaches.

4.5.2 Results when applying soft decision functions

We begin using *soft-decision* functions with method 4 from the figure 4.4. This method gives the best performance in the *target* task but at the expense of a unsatisfactory result in the *source* task (close to what was obtained with the fine-tuning method). Because of how slow this function decreases, many neurons are assigned to large weights, giving a lot of room to the network to learn by modifying a higher number of neurons.

Next method uses a linear function (method 5 in figure 4.4) which decreases faster than the previous experiment function. Method 5 results in a similar performance for the *target* task and an improvement in the *source* one (around 5% better compared to method 4 for both experiments). Even so, it does not approach the results obtained for the *source* task with methods 2 and 3, so the next step will be to apply a function with a more aggressive descent.

Last method studied (method 6 in figure 4.4) consists in applying a function with

a high initial slope expecting the CNN to have less freedom so it maintains a good performance in the *source* task. The changes in the *target* task performance are small, and the *source* task performance increase compared to the method 4 and 5. However, these results are still worse than those obtained in experiments 2 and 3.

4.5.3 Final discussion

After examining the results, method 2 stands out among the others for being the one that presents less joint loss. Same statement can be clearly seen in the figure 4.6 for the first experiment and in the figure 4.8 for the second one. This behaviour indicates that 15% of unconstrained neurons is enough for the network to learn the explored *target* tasks without losing excessive performance in the *source* tasks.

Also, for both experiments *hard-decision* transfer functions provide better performance than *soft-decision* transfer functions.

Finally, in the light of the results, all the methods presented here achieve a better result than the ones obtained with the fine-tuning method. This confirms the initial hypothesis of the work: the value H of a neuron is an indicator of its relevance in the task and can be used to design learning rates that allows a CNN to learn a new *target* task without forgetting an initial *source* task.

Chapter 5

Conclusions and future work

5.1 Conclusions

A visualization tool has been adapted to the needs of this project, adding some modules that were considered necessary. By using this tool, we have understood the activation patterns of neurons in a CNN. Next, we have designed a quantitative measure, known as H, to estimate these neurons activations along one data-set.

Then we have developed different methods to apply this measure to the individual learning rates of the neurons. Last, we have evaluated these methods through different experiments, obtaining significant better results than the ones obtained by the fine-tuning strategy.

In conclusion this is an experimental work that confirms our initial hypothesis and paves the road towards further exploration of alternative activation measures and weighting functions.

5.2 Future work

In the light of the results, we proposed three lines of future work:

- Asses the feasibility of the proposed method by exploring alternative *source* and *target* tasks.
- Experiment with different transfer functions and compare their performance with the ones describe in this document.
- Design alternative measures to estimate the neurons' activations and evaluate its performance.

Bibliography

[1] A. Rannen, R. Aljundi, M. B. Blaschko, and T. Tuytelaars, "Encoder based lifelong learning," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1320–1328, 2017.

[2] R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars, "Memory aware synapses: Learning what (not) to forget," in *Proceedings of the European Conference on Computer Vision*, pp. 139–154, 2018.

[3] "VisualDL - Visualize your deep learning training and data flawlessly." http://visualdl.paddlepaddle.org/, 2018. Accessed: 2019-02-25.

[4] "Matlab - Deep Learning Toolbox." https://es.mathworks.com/products/deep-learning.html. Accessed: 2019-02-12.

[5] L. Berrada, A. Zisserman, and M. P. Kumar, "Deep frank-wolfe for neural network optimization," *Computing Research Repository*, 2018.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, pp. 1097–1105, 2012.

[7] "Open neural network exchange.." https://github.com/onnx/onnx. Accesed: 2019-02-10.

[8] A. W. Harley, "An interactive node-link visualization of convolutional neural networks," in *Procedings of the International Symposium on Visual Computing*, pp. 867–877, 2015.

[9] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database. 2010," *http://yann. lecun. com/exdb/mnist*, 2010.

[10] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson, "Understanding neural networks through deep visualization(2015)." https://github.com/yosinski/deep-visualization-toolbox. Accesed: 2019-03-12.

[11] Arikpoz, "deep-visualization-toolbox modified." https://github.com/arikpoz/deep-visualization-toolbox. Accesed: 2019-03-13.

[12] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Neural networks for perception*, pp. 65–93, 1992.

[13] L. Torrey and J. Shavlik, "Transfer learning," in *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pp. 242–264, 2010.

[14] A. R. Zamir, A. Sax, W. Shen, L. J. Guibas, J. Malik, and S. Savarese, "Taskonomy: Disentangling task transfer learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3712–3722, 2018.

[15] A. Rannen, R. Aljundi, M. B. Blaschko, and T. Tuytelaars, "Encoder based lifelong learning," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1320–1328, 2017.

[16] R. Aljundi, P. Chakravarty, and T. Tuytelaars, "Expert gate: Lifelong learning with a network of experts," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3366–3375, 2017.

[17] R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars, "Memory aware synapses: Learning what (not) to forget," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 139–154, 2018.

[18] A. Quattoni and A. Torralba, "Recognizing indoor scenes," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 413–420, June 2009.

[19] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[20] xxrandon, "Pytorchtocaffe." https://github.com/xxradon/PytorchToCaffe, 2018. Accesed: 2019-04-28.